InfoDB

Fall 1990

Volume 5 Number 3

The Leading Journal for Database Users

In this issue ...

[illegible]

Database directions.

[illegible]

As applications become more complex, the quality of a relational DBMS's optimizer plays an increasingly more important role in achieving good performance. This article looks at the features of an optimizer, and reviews the optimizers of several commercial products.

[illegible]

In this installment of his regular contribution to *InfoDB*, Chris Date looks the subject of composite keys in database design. He discusses when they should be avoided, and when they are useful.

[illegible]

A year ago, we reviewed the IBM announcement of its AD/Cycle strategy. Since then, IBM has released the Repository Manager component of AD/Cycle, and has revealed details of the Information Model it is developing with its business partners. We look here at the progress IBM has made over the past year, and review the issues facing potential users of the Repository and its associated CASE tools.

[illegible]

In this interview, Paul Winsberg discusses the status of AD/Cycle with Judy Teson, Manager of Market Strategy and Support at IBM's Santa Teresa Lab.

[illegible]

As the industry moves towards the use of distributed processing and distributed database, there is increasing need to maintain multiple copies of the same data at multiple locations. One effective way of doing this is via a snapshot facility. This article looks at the uses of snapshots, and presents the features required by such a facility.

[illegible]

Database applications are becoming more complex, and many database users are finding that the classical method of controlling concurrent access to data using a locking protocol is creating performance problems. Here, we look at this issue, and review the strengths and weaknesses of some vendor solutions to this problem.

Relational DBMS Optimizers: An Evaluation

Nothing can help more in estimating the performance of a Relational Database Management System (RDBMS) than a thorough understanding of its optimizer. The optimizer also determines a good deal about a product's functionality — for example, is the DBMS appropriate for distributed database management, *ad hoc* query processing, batch processing, on-line transaction processing,[15] on-line complex processing,[16] or parallel processor exploitation? This article discusses the features of an optimizer and reviews the optimizers of several commercial products.

by David McGoveran

Evaluating and classifying optimizers is a difficult task. There is probably no linear measure of how good an optimizer is, because different applications will require different optimizer features. In some cases, features that may be absolutely necessary for one DBMS would be irrelevant for others. If, for example, a DBMS supports only one access method, then there is no need for the optimizer to be capable of evaluating and selecting between multiple access methods.

In a recent article,[10] I explained some of the workings of optimizers, listed some features to look for when evaluating an optimizer, and proposed a simple scheme for classifying a DBMS based on the features of its optimizer. This article extends that work and starts the process of applying it. The goal is to develop a classification scheme and an empirical method of evaluating optimizers.

In this article we review some of the features which an optimizer might have, and look at how these features are supported by nine products:

- CA-DB (Release 1.4)
- DB2 (Version 2 Release 2)
- Informix OnLine (Release 4.0)
- Ingres (Release 6.3)
- Oracle (Release 6.0)
- Rdb/VMS (Release 4.0)
- ShareBase III
- Sybase SQL Server (Release 4.0)
- Tandem NonStop SQL (Release 2)

Identifying Optimizer Characteristics

The optimizer's job is to produce an optimal processing strategy or *query execution plan*. As mentioned above, a set of objectives and features has already been developed[10] for evaluating how efficient an optimizer is at this task. This article discusses this evaluation scheme in more detail. Note: The list of subfeatures under each of the numbered features in the evaluation scheme below is not necessarily exhaustive. It is for the user to judge how many of these subfeatures must be supported by a product to satisfy a given feature.

Table 1 summarizes how the nine products listed above satisfy the list of features in the evaluation scheme. The summary is taken from Reference 19, which documents in more detail the support provided by each product.

General Features

1. Can the optimizer both interpret and compile plans at the user's discretion?

- optional interpretation and compilation/automatic recompilation
- compiled only
- interpretive only

Interpretive systems are useful in *ad hoc* and dynamic Data Manipulation Language (DML) applications, or perhaps where the database schema or data population changes frequently. Compiled systems are efficient where the DML is known in advance of execution time and is not expected to change. Some "interpreted systems" are highly efficient and use pre-existing optimized code (such as stored procedures[18]), so that the line between interpreted and compiled systems is not as clear as it once was. In addition, it is possible for some interpreted systems to save their output as compiled code for later reuse. Be aware that there is a spectrum between pure compiled versus pure interpreted systems.

2. Does the optimizer use cost functions as compared to the less desirable cost indexes or general cost heuristics?

- uses cost functions
- uses cost indexes
- uses general cost heuristics

Cost functions are used to estimate costs based on estimates of the amount of data to be manipulated by each operation — they generally use other statistical factors in each cost function as well. *Cost indexes* are much simpler — they just add an incremental cost for each type of operation when producing a cost estimate for a plan. Cost indexes may also be used as simple weighting factors and can be multiplied by the amount of data estimated to be manipulated by each operation. *Cost heuristics* involve selecting operations for execution based on a precedence list of the relative value of each operation. The operations may be high-level and based on DML expression or predicate syntax. Typically there is no quantitative value associated with a processing strategy selected on the basis of cost heuristics.

3. Is the optimizer insensitive to syntactic variations?

- order of tables
- order of columns in the WHERE clause
- order of expressions/predicates
- subquery versus join
- EXISTS versus IN
- COUNT versus EXISTS

Sensitivity to the phrasing of an SQL query places a burden on users who are concerned with performance. On the other hand, it also allows the sophisticated user to "manually" optimize the query. Syntax sensitivity is removed by conversion of an SQL statement to a *canonical form* and perhaps by *flattening* as well, so that phrases that are logically the same, except for syntax, are optimized in the same way. I have previously refered to an optimizer which determines the optimal plan according to the particular syntax used, as *syntax-based*, but I now prefer *syntax-driven* or *syntax-sensitive* as more specific variants.

4. Can the optimizer perform a semantic transformation?

If an RDBMS supports integrity constraints, it is possible to automatically transform *semantically* equivalent (as compared to *syntactically* equivalent) queries one into the other, or to infer additional restrictions and join conditions from the integrity constraints. This can result in simplifying a query in ways that converting to canonical form and flattening cannot. This improves and gives more consistent performance for queries that mean the same thing, regardless of how they are phrased.

5. Does the optimizer recognize and use various "logical laws," i.e., does it do syntax transformation?

- transitivity of equi-joins and theta-joins
- equivalence of: *not greater (less) than* and *less (greater) than or equal to*
- DeMorgan's Laws — equivalence of: *not (a and b)* and *not a or not b*; equivalence of: *not (a or b)* and *not a and not b*
- associativity
- commutativity
- distributivity

If a WHERE clause contains equi-joins $a=b$ and $b=c$, and there are indexes on columns a and c, but not on column b, the join $a=c$ is implied and can be performed using the indexes on a and c. This is likely to reduce the number of rows that must be joined to b. Other useful laws include DeMorgan's Laws, the equivalence of *not greater than* and *less than or equal to*, etc. Both scalar and relational operations should be considered. Ideally, users should not have to be logicians. For a good list of useful tautologies, see Reference 14.

6. Does the optimizer optimize over all the standard predicate types and special extensions like data type conversion functions?

- AND, OR, IN, UNION, BETWEEN, LIKE, NOT, NULL,
- subqueries and correlated subqueries
- data type conversion functions
- scalar and aggregate functions

For example, various optimizers fail to optimize predicates involving one or more of OR, UNION, IN, BETWEEN, LIKE, NOT, NULL, subqueries, correlated subqueries, functions, etc. The more of these supported, the greater the power of the language and its utility in mission critical applications.

7. Is there a powerful EXPLAIN-like facility and means for using the information it provides?

- shows the internal or coded format of a plan
- shows plans considered, but not selected
- shows graphical display of a plan
- shows textual explanation of a plan
- has direct means to influence the optimizer (i.e., the user can select a particular plan, or can specify the order in which tables are processed and the indexes to be used)
- has the indirect means to influence optimizer (e.g., by setting statistics)

An EXPLAIN facility allows the user to obtain a description of the access plan that the optimizer has selected for the query. This can be extremely useful in performance optimization if the user has some means to influence the optimizer. Some DBMSs provide a direct means of influence (e.g., telling the optimizer which access plan to use) and others an indirect means (e.g., modifying the syntax of the query or modifying indices). These features are needed in production MIS shops.

8. Is the optimizer capable of handling arbitrarily complex queries (i.e. what are its limitations)?

- maximum number of tables
- maximum number of joins
- maximum number of subqueries
- maximum depth of nested subqueries
- maximum query size
- maximum number of columns in a query or SELECT list

For example, it is not uncommon to see the number of tables allowed in a query restricted to sixteen. However, some optimizers effectively "give up" when the number of tables in a join is as few as five or six. For OLCP and *ad hoc* decision support applications, this is unacceptable.

Access Method Support

9. Is the optimizer capable of manipulating the order of operations?

The order in which SQL operations are executed can determine the size of intermediate results. If the optimizer does not attempt to compute or keep track of intermediate results, it can not evaluate the order dependence of operations. The correctness of cost estimate results and the selected plan is, therefore, suspect.

10. Does the optimizer evaluate the various join methods available effectively?

- supports multiple join methods
- evaluates multiple join methods

The difference in cost of sort-merge, index only, nested loop, and partitioned hash join methods can be considerable. If the optimizer treats all joins equally or fails to evaluate a particular method properly, anomalous performance behavior results.

11. Does the optimizer select an appropriate sorting algorithm?

- supports multiple sorting algorithms
- evaluates cost of sorting algorithms

Each sorting algorithm has advantages and disadvantages. For example, a Quicksort becomes embarrassingly slow when faced with already sorted data. If the optimizer uses a single sorting algorithm, it should be one for which such anomalous results do not occur, or the conditions should be recognized so that the algorithm will not be applied inappropriately.

12. Does the optimizer recognize inter- and intra-table clustering in evaluating the cost of a query?

- inter-table clustering
- intra-table clustering

The optimizer should, for example, recognize the cost advantage of intra-table clustering (i.e. the optimization of physical storage record order for a single table) for sorted retrievals or range queries. Similarly, it should recognize that inter-table clustering (i.e., storing rows from two or more tables physically together) may be advantageous when a primary key join on the clustered tables is required. It is a disadvantage, however, when only one of the clustered tables is accessed.

13. Does the optimizer take I/O bandwidth into account?

Not all disk drives are created equal, and, therefore, some means of factoring in disk drive performance is advantageous, preferably at configuration time. The administrator might put rotational speed, latency, and mean transfer rate in a table for use by the optimizer whenever a drive or other storage medium is added to the system, or the system might monitor drive performance automatically. For an extreme example, consider the ineffectual use of a RAM disk drive if it is treated as a standard disk drive.

14. Does the optimizer evaluate the advantage or cost of buffering?

- recognizes available buffer space
- differentiates input/output/intermediate processing buffer space
- evaluates cache/buffer management costs
- takes disk paging into account

All systems have limited buffer space. If the optimizer assumes effectively unlimited buffer space, the cost estimate will sometimes be too low due to paging. On the other hand, if it ignores the possibility of caching and buffer management, the cost estimates will be too high. The cost function should represent the buffer management algorithms available.

15. Does the optimizer take into account the costs of transaction management, journaling, consistency enforcement, and concurrency?

- transaction management
- journaling
- consistency enforcement
- concurrency

The costs of setting and resetting locks, and of resource waits will contribute to the overall cost of executing a query. Statistics such as the probability of deadlocks, average resource wait, etc., based on number of concurrent users should be factored in. In most cases, these costs are constant for all plans. However, for distributed optimization, and for optimizers that optimize across statements within a transaction, the order of these operations can affect the overall cost. These issues are important in OLCP and OLTP applications.

Index Support

16. Does the optimizer use different kinds of indices effectively?

- uses different types of indices
- evaluates plan cost by type of index

Various indexing methods exist, each of which has different uses (and may have a different storage structure). For example, a hash index is better for single record access, while a B-tree index is better for finding ranges of values. Indices can be made ineffectual by poor optimizer evaluation, and considerable care in index design will be required to compensate for this.

17. Does the optimizer recognize potentially useful indices in all reasonable cases?

- when a leading partial index value is specified
- when a trailing partial index value is specified
- when *not equal to* a partial index value is specified
- when (part of) an index value occurs in any function
- when (part of) an index value occurs in an aggregate function
- when (part of) an index value occurs in an invertible function

An optimizer might not always recognize the usefulness of an index. If a composite index is defined on three columns of a table, and only the first two are mentioned in a query, the optimizer may or may not recognize that the columns form part of an index. If a product supports breadth first searching of an index, then the leading columns of a composite index need not be known for the index to be useful. Some optimizers do not recognize a column as belonging to an index if it is used in any computation or a function. Others will not use indices on a column if the column occurs in certain kinds of comparison operations such as "not equal" — if "not equal" serves to exclude all data except that in a narrow range or ranges of values of an indexed column, the index can be used to eliminate an unnecessary relation scan.

18. Can the optimizer use or at least deal with multiple indices?

- uses multiple indices to reduce data I/O
- uses multiple indices to perform an index-only join
- uses at least one index if useful, regardless of how many exist

An effective optimizer will not only use multiple indices, but will also attempt to perform an "index join," "union," or "intersect" where possible. The idea is to access only those pages referenced by all relevant indices.

19. Can the optimizer create useful indices automatically?

- creates an index on large intermediate results
- may create a temporary index if useful
- may create a permanent index if useful

When a potentially useful index does not exist, some optimizers will create one, assuming that the cost of creating the index is less than processing the alternate plans. Typically, this will be a temporary index, disappearing after the statement is processed. However, it is possible that knowledge of usage patterns may be used in deciding the cost of creating a permanent index.

For example, heavily used stored procedures or repeated queries in a read intensive environment may benefit from such indices. Similarly, sorting or creating temporary indices on intermediate results may be beneficial. This can be important in processing very large base tables, and in OLCP, batch, and decision support applications where large results tables are more common.

20. Is the optimizer capable of using multi-table indices?

- supports multi-table indices
- takes multi-table indices into account

Some systems provide a mechanism for creating a single index on the columns in multiple tables. This improves join performance and can be used for referential integrity enforcement. The optimizer should not only recognize when these indices are useful, but should be able to use the index for the lookup of keys from either table.

21. Can the optimizer respond to user created index and access methods?

- supports user created or specified access methods
- supports user created or specified index methods
- takes user created or specified access methods into account
- takes user created or specified index methods into account

A few RDBMSs allow the user to specify index methods and access methods as user exits (e.g., Ingres, Starburst). These are useful in creating foreign DBMS gateways, and for dealing with applications with special data (e.g., CAD). It is important that the optimizer recognize these extensions. If it does not, they are of little use unless performance is unimportant.

22. Does the optimizer handle nulls properly?

- index can be used even if a column can contain NULLs
- optimizes IS NULL and IS NOT NULL restrictions
- optimizer recognizes how NULLs affect transformations (i.e., it understands when to apply three-valued logic)

Some optimizers refuse to use any index (not just the primary key index) on a column that can contain nulls. Other optimizers can not optimize a restriction that involves NULLS (i.e., IS NULL and IS NOT NULL). A preferred approach is for the optimizer to recognize when it can use the richer two-valued logic and when it must restrict optimization techniques to those allowed by three-valued logic.

Statistics

23. Is the optimizer sensitive to table and index statistics?

- cardinality (number of distinct table rows)
- rows per physical page
- number of data pages for a table
- percentage of total pages occupied by the table
- number of index pages
- index selectivity

Some products keep track of the number of rows in a table. Others record the average number of rows per page, number of pages per table, percentage of total pages, and the number of index pages. The selectivity of an index is also important. This statistic tells the optimizer the number of rows in a table for each value in the index. Some statistical optimizers rely entirely on these statistics for optimization.

24. Does the optimizer keep track of data value distributions?

- user-managed histograms
- automatic histograms
- multi-modal distributions
- skewed distributions
- maximum column value
- minimum column value
- average column value
- avoids false maximurn/minimum values

The cardinality of a table, and the maximum, minimum, and average value for a column are useful for heuristic measures. A few database products maintain fairly sophisticated statistics about the distribution of data values in a table. This allows the optimizer to analyze the number of disk I/Os required to satisfy a query more effectively. If the optimizer assumes a uniform distribution of data values across all table pages, but the distribution is in fact skewed or multi-modal, then the actual number of pages required to access data can be very different (either higher or lower) than the estimate. Keeping minimum and maximum values is not sufficient to distinguish between uniform and non-uniform distributions. Keeping just the minimum and maximum value of a column can also give false information to the optimizer, if nulls are stored as the highest or lowest value.

25. Can the optimizer estimate the number of disk pages that must be accessed?

- uses a cost function
- uses a cost index
- uses a general heuristic

The cost of disk I/O can not be computed if the optimizer can not estimate the number of data pages that must be accessed, i.e., it can not use a cost function. Optimizers that do not have this capability typically use a cost index, rather than a cost function, to estimate disk I/O. A cost index is a numeric value given to a particular operation, regardless of the amount of data that must be processed. The sum may then be scaled according to the amount of data. The result is optimization based on gross statistical or theoretical assumptions. This is similar to measuring the health of the U.S. economy based on the Dow Jones Industrials (an economic index) instead of computing the GNP, trade deficit, and inflation.

26. Does the optimizer provide a mechanism for updating statistics?

- provides automatic updates
- provides means for on demand updates

- provides triggered updates (e.g., when tables change size by a certain percentage)
- continuous updates
- scheduled updates
- requires manual updates

Optimizers can update statistics continuously, automatically, manually, on a scheduled interval, by a trigger (e.g., when new extents are allocated, new indices are created, etc.). If manual updating of statistics is allowed it should be handled through a utility that protects the user from serious errors. Some optimizers allow the updating of statistics to be restricted to a specific table or column. Others permit the DBA to directly update statistics using DML operations. Both of these options are dangerous and can lead to problems. The most appropriate method for updating statistics will depend largely on usage patterns and data value distributions.

27. Does the user have controls to manage the cost of obtaining statistics from the database?

- sampling can be used
- a sample database can be used
- uses continuous update of cached statistics
- uses continuous update of non-cached statistics
- uses table scan

When a database is very large, the cost of updating the statistics can become exorbitant. One solution to this problem is to produce the statistics by sampling the data. A variation is to generate the statistics from a sample or test database. Similarly, statistics can be updated continuously, either for all activity on a database or on a random sampling basis. Up-to-date statistics are critical for performance, particularly in a distributed DBMS environment.

Efficiency Features

28. Can access plans be saved or cached?

- for the duration of a transaction
- for the duration of a program or process
- until the DBMS is restarted
- permanently
- manually (i.e., at user request)
- automatically

Access plans can be cached for repeated queries, eliminating the overhead of generating an access plan when an application uses the same query repeatedly. Similarly, database procedures generate access plans which are stored in the database on the first invocation. The optimizer should provide a means for checking the validity of stored or cached plans, and should be capable of regenerating a plan automatically if need be.

29. Can the optimizer recognize and re-use parts of an existing plan?

- within a statement
- within a transaction
- within an application
- belonging to the same user process
- across applications and users
- across cached plans
- across stored plans

The optimizer might maintain a cache of the plans it has recently executed. If a new plan needs to be selected, it can try to find an existing equivalent plan. Failing this, it should look for parts of the plan which have already been computed and optimized.

30. Does the optimizer recognize invalid plans or invalid partial plans?

- plans
- parts of plans
- when indexes are added
- when indexes are dropped
- when statistics change significantly
- when a table is dropped
- on user request

This is particularly important for compiled systems. Changes to the statistics or to the database schema can result in a plan (or partial plan) becoming invalid. Likewise, a minor variation on a plan (or partial plan) may make it invalid, for example, when substituting a wildcard for a constant in a boolean comparison search condition.

31. Does the optimizer optimize aggregate functions?
Aggregate functions occur frequently in OLCP and decision support applications. There are several techniques for optimizing aggregate functions. For example, an "unnesting technique" can be used to separate the aggregate function from the rest of the statement. Index-only access is also possible when evaluating some aggregate functions.

32. Does the optimizer recognize repeated occurrences of an aggregate function?
If an aggregate function occurs repeatedly in a query, or if the arguments to different aggregate functions (e.g., *sum(x)* and *avg(x)*) recur, they need be evaluated only once. This means that the optimizer must recognize the repeated occurrence of potentially complex subtrees in the plan, and cause them to be evaluated only once.

33. Is dynamic selection of plans supported at runtime?
If multiple plans are generated for the same query, the optimizer can switch at runtime between plans based on a maximum allowed real cost (or switching threshold) for a particular plan. In this way, if the real cost at runtime exceeds the switching threshold, the optimizer can switch to a new, potentially better plan. It is also possible for the plan switching to be based on the runtime values of dynamic SQL parameters.

34. Can the optimizer optimize transactions as well as statements?

- statements (*local* optimization)
- transactions (*global* optimization)

If it is possible for the optimizer to look ahead and see all the statements in a transaction (as may be the case in a database procedure or stored procedure), it may be possible to cache intermediate results for use later in the same transaction, or to minimize the amount of disk I/O by scanning a clustered table, even though some of the data will be used only by later statements in the transaction. Some of the effect of "global" optimization can be obtained if intermediate results are cached routinely and the optimizer is capable of caching and recognizing partial plans used by previous statements. This type of optimization could also be done for parts of a transaction, rather than the whole transaction.

	CA-DB	DB2	INFORMIX	INGRES	ORACLE	RDB	SHAREBASE	SYBASE	TANDEM
General Features									
1.	Y	compiled	interpreted	Y	interpreted	interpreted	Y	Y	compiled
2.	Y	Y	Y	Y	N(heuristics)	Y	Y	Y	Y
3.	Y	P	Y	Y	N	P	Y	N	P
4.	Y	N	N	P	N	P	Y	N	P
5.	Y	P	P	P	P	Y	Y	P	P
6.	Y	Y	Y	Y	P	Y	Y	P	P
7.	P	Y	Y	Y	P	Y	Y	Y	Y
8.	Y	Y	Y	Y	Y	Y	Y	Y	Y
Access Method Support									
9.	Y	Y	Y	Y	Y	Y	Y	Y	Y
10.	Y	Y	N	Y	Y	Y	Y	Y	Y
11.	Y	N	N	N	P	Y	N	N	Y
12.	P	P	P	P	P	Y	P	P	P
13.	Y	N	N	N	N	N	P	N	N
14.	Y	Y	Y	Y	N	Y	Y	Y	N
15.	Y	N	N	N	N	N	N	N	N
Index Support									
16.	Y	N	P	Y	Y	Y	P	N	N
17.	P	P	P	Y	P	Y	Y	P	P
18.	Y	Y	P	Y	Y	Y	P	Y	N
19.	N	N	Y	Y	N	P	Y	P	P
20.	Y	N	N	N	Y	N	N	N	N
21.	N	N	N	N	N	N	N	N	N
22.	Y	Y	Y	Y	N	Y	Y	Y	Y
Statistics Support									
23.	Y	Y	Y	Y	N	Y	Y	Y	Y
24.	P	P	N	Y	N	P	N	Y	N
25.	Y	Y	Y	Y	N	Y	Y	Y	Y
26.	Y	Y	Y	Y	N	Y	Y	Y	Y
27.	Y	P	N	Y	N	P	P	N	P
Efficiency Features									
28.	P	P	N	Y	P	N	Y	Y	P
29.	N	N	N	Y	Y	N	N	N	N
30.	P	P	P	Y	N	P	P	P	P
31.	Y	Y	Y	Y	Y	Y	Y	Y	Y
32.	Y	Y	N	Y	Y	P	Y	Y	N
33.	N	P	N	N	N	Y	N	N	N
34.	N	N	N	N	N	N	N	N	N
Artificial Intelligence Features									
35.	Y	Y	Y	Y	P	Y	Y	Y	Y
36.	N	N	N	N	N	N	N	N	N
37.	P	N	Y	Y	N	P	N	N	P
38.	N	N	Y	Y	N	N	N	N	N
39.	N	N	N	Y	N	N	N	N	N
40.	P	N	N	Y	N	P	P	N	Y
41.	P	N	N	Y	N	N	P	N	Y
42.	Y	N	N	Y	N	N	N	N	N

Table 1. Comparison of nine commercial optimizers

Artificial Intelligence Features

35. Does the optimizer use heuristics to eliminate plans?

- prior to generating all possible plans
- prior to evaluating all possible plans

Heuristics can be used to eliminate plans prior to cost function evaluation. During cost function evaluation, heuristics can also be used to terminate the evaluation of the entire plan. These techniques allow optimizer time to be spent examining potentially more useful plans. Note that this is different from using heuristics for cost estimates.

36. Can the optimizer learn?

- from usage patterns
- from actual performance measurements

In some minimal sense, an optimizer learns if it collects and responds to statistics. Likewise, it also learns if it uses partial plans from previous optimization efforts. However, stronger forms of learning are possible. For example, the optimizer could compare actual and estimated costs to adjust its cost estimate algorithms. Search heuristics could be used to reduce costs — see Pierls, Reference 12. Usage patterns could be used to dictate the spreading of data across devices or even across data pages. The frequency and amount of data updating could dictate the optimal extent size when new pages need to be allocated. The frequent access of a column could lead to automatic creation of indices. Automatic reorganization of the data is possible when a particular processing order is common, or when fragmentation exceeds a cost threshold. Another potentially fruitful area is the use of *genetic algorithms* — see Reference 17.

Parallel Processing and Distributed Support

37. Is the optimizer capable of computing distributed cost functions?

- routing
- network bandwidth
- node CPU speed
- parallel query processing
- parallel disk I/O

When data is distributed, factors such as the routing of the retrieved data, network bandwidth, node CPU speed, and possible concurrent or parallel processing of the decomposed query, all become important in computing costs. These factors, as well as the normal local statistics must be available to the global optimizer. How these statistics are sent from the local database to the global optimizer is also important (e.g., by manual update, automatic update, on demand, or triggered by an event such as storage space allocation, etc.).

38. Does the optimizer address both local and global factors?

- local (single node factors)
- global (multiple node factors)
- both

Even if the optimizer performs global optimization in a distributed environment, it should still optimize local queries, and not depend on a plan dictated by the global phase. If the optimizer does both local and global optimization (as with Ingres/Star, for example), the method by which these phases interact (i.e., which is done first, and how the costs are propagated between the phases), and the method by which statistics are updated are both important. Note that this use of the terms local and global is different from that used in question 34 above.

39. Can the user control how the global optimizer obtains access to local statistics?

- manually (separate command or utility)
- on demand
- on a user-defined schedule
- statistics are replicated on each node

Access to local statistics by a global optimizer could be managed manually, on demand, or on a user-defined schedule. Global optimization requires not only information about where data resides, and how best to route the necessary data to the user, but also about how best to prepare that data for distributed access in the first place.

40. Can the optimizer evaluate parallel I/O?

- for replication
- for fragmentation

If replication and fragmentation are supported by the RDBMS, the optimizer might be able to use multiple disk drives and controllers to process the results in parallel. It is not necessary for the RDBMS to be distributed for this to be a benefit.

41. Can the optimizer take advantage of parallel processing?

- differentiates between parallel and sequential CPU costs
- adapts to system configuration changes

In a distributed system, which portions of a query are processed in parallel, and which must be processed sequentially, can have a major effect on the processing cost. If the optimizer does not model the system properly, parallel processing may be more costly than sequential processing on a uniprocessor machine. Processing may be done in parallel within query operations or across query operations. Again, it is not necessary for the RDBMS to be distributed for this to be a benefit, but multiple CPUs need to be available to the query processor.

42. Does the optimizer compute the cost of semijoins?

Semijoins are sometimes an effective means of performing joins where the data is distributed and network costs are significant. The cost of a semijoin should not be evaluated like that of a join.

Classifying the Optimizer

When attempting to evaluate an optimizer, it is convenient to classify the DBMS according to the type of optimizer provided. A scheme that I find useful is as follows:

Class 0: There is no optimizer.

Class 1: The optimizer is syntax driven or sensitive. It does not use statistics and uses either cost indices or heuristics to determine the access plan.

Class 2: The optimizer is syntax sensitive, but does use cost functions. It does not use statistics except perhaps to estimate table size.

Class 3: The optimizer is syntax sensitive, uses cost functions, and uses minimal statistics.

Class 4: The optimizer is not syntax sensitive, uses cost functions and statistics. Facilities to influence the optimizer manually may be available.

Class 5: The optimizer is not syntax sensitive, uses cost functions, and uses extensive statistics, including those about the distribution of data values. The user may be able to limit the search cost or to force an exhaustive search.

Class 6: The optimizer is not syntax sensitive, uses cost functions, extensive statistics, and performs global optimization in a distributed database environment.

These six classes are meant as a guide, no more. The classification assumes that performance should not be influenced by syntax, that cost functions are better than cost indices or simple heuristics, that statistics are useful, and that user facilities for controlling the optimizer are beneficial.

It is important to understand that a particular optimizer may have some characteristics found in a higher class. Similarly, an optimizer in a higher class may still use the techniques found in a lower class optimizer, but will do so judiciously. It may be that certain characteristics, while not rated highly by the classification, will be acceptable nonetheless. For example, the fact that a particular DBMS optimizes only at compile time (as distinct from run or execution time) can justify its use of exhaustive search. If the optimizer is then able to re-use access plans or partial plans, this technique may not be particularly detrimental, even in an *ad hoc* environment. Similarly, if the users of the DBMS are sophisticated and willing to keep in mind possible performance penalties from poorly phrased SQL, syntax sensitive optimizers may be acceptable.

Acknowledgements

The author would like to thank Chris Date, Colin White, Eric Lundblatt, Mike Pong, Paul Bass, Ken Jacobs, Chuck Kelly, Yogesh Gupta, Howard Torf, and Gilbert Wai for their help with this article.

References

1. Gabrielle Wiorkowski and David Kull. "The Optimizer: Invisible Hand of the DBMS." *Database Programming and Design*, September 1988, Volume 1, Number 9.
2. E.F. Codd. "Fatal Flaws in SQL (Part 1)." *Datamation*, August 1988.
3. E.F. Codd. "Fatal Flaws in SQL (Part 2)." *Datamation*, September 1988.
4. Dave Kellogg. "Understanding Query Optimizers." Relational Technology Technical Report, July, 1989.
5. Robert Kooi. "The Optimization of Queries in Relational Databases." Ph.D. dissertation, Case Western Reserve University, September 1980.
6. W. Kim, D. Reiner, and D. Batory (Ed). *Query Processing in Database Systems*. Springer Verlag, 1985.
7. C.J. Date. *An Introduction to Database Systems, Volume 1, Fifth Edition*. Addison-Wesley, 1990.
8. M. Stonebraker (ed). *Reading in Database Systems* — Chapter 2. Morgan Kauffman Publishers, 1988.
9. C.J. Date. "Be Careful with SQL EXISTS!" *Database Programming and Design*, September 1989, Volume 2, Number 9.
10. David McGoveran. "Evaluating Optimizers." *Database Programming and Design*, January 1990, Volume 3, Number 1.
11. Ole Jorgen Anfindsen. *A Study of Access Path Selection in DB2*. Norwegian Telecommunications Administration, October 1989.
12. J. Pearl. *Heuristics: Intelligent Search Strategies For Computer Problem Solving*. Addison-Wesley, 1984.
13. J.J. King. "QUIST: A System for Semantic Query Optimization in Relational Databases." Proc. 7th International VLDB Conference, Cannes, France, September, 1981.
14. P. Suppes. Page 34 in *Introduction to Logic*. Van Nostrand Rheinhold, 1957.
15. Colin White. "What is OLTP?" *InfoDB*, Volume 4 Number 1, Spring, 1989.
16. David McGoveran. "Beyond OLTP: On-Line Complex Processing." *InfoDB*, Volume 5, Number 1, Spring/Summer 1990.
17. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
18. David McGoveran. "The Power of Stored Procedures." *Database Programming and Design*, Sept. 1989, Volume 2, Number 9.
19. *RDBMS Optimizer Evaluation*. Database Evaluation Report Series, Database Associates, 1990.

David McGoveran is a Principal of Database Associates and an Associate Editor of *InfoDB*. He is also President of Alternative Technologies, a database development company based in Boulder Creek, California.